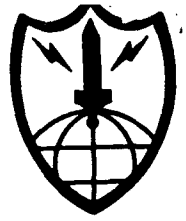


USAISEC

US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

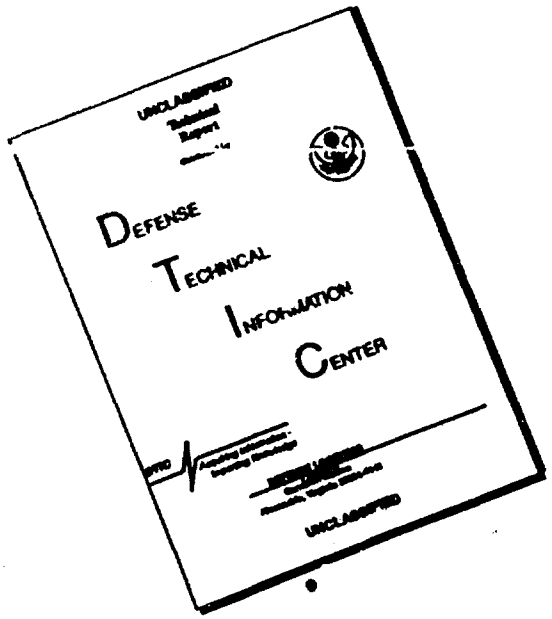
Evaluation of DCDS for Meeting the Data Collection Requirements for Software Specification, Development, and Support

February 1991

ASQB-GI-[illegible]

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. [illegible] Exp. Date [illegible]

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b RESTRICTIVE MARKINGS: NONE
2a SECURITY CLASSIFICATION AUTHORITY: N/A	3 DISTRIBUTION / AVAILABILITY OF REPORT: N/A
2b DECLASSIFICATION / DOWNGRADING SCHEDULE: N/A	
4 PERFORMING ORGANIZATION REPORT NUMBER(S): ASQB-GI-91-009	5 MONITORING ORGANIZATION REPORT NUMBER(S): N/A
6a NAME OF PERFORMING ORGANIZATION: Purdue University / SERC; 6b OFFICE SYMBOL (if applicable):	7a NAME OF MONITORING ORGANIZATION: N/A
6c ADDRESS (City, State, and ZIP Code): Department of Computer Science, West LaFayette, Indiana 47907	7b ADDRESS (City, State, and ZIP Code): N/A
8a NAME OF FUNDING SPONSORING ORGANIZATION: AIRMICS; 8b OFFICE SYMBOL (if applicable): ASQB - GI	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER:
8c ADDRESS (City, State, and ZIP Code): 115 O'Keefe Bldg., Georgia Institute of Technology, Atlanta, GA 30332-0800	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO: 62783A; PROJECT NO: DY10; TASK NO: 02-04-02; WORK UNIT ACCESSION NO:

11 TITLE (Include Security Classification): Evaluation of DCDS for Meeting the Data Collection Requirements for Software Specification, Development, and Support (UNCLASSIFIED)

12 PERSONAL AUTHOR(S): Dunsmore, Buster, Varnau, Steve

13a TYPE OF REPORT: final report	13b TIME COVERED: FROM ____ TO ____	14 DATE OF REPORT (Year, Month, Day): 1991, February, 12	15 PAGE COUNT: 77

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	CASE Tools, Distributed Computing Design System, DCDS, Teamwork, Excelerator, FPOS, DesignAid, SA Tools, Software engineering environment Systems, SEES, Ada Programming Support Environment, APSE

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This technical report consists of five separate but related reports which evaluate the Distributed Computing Design System (DCDS). DCDS was developed by TRW as a software development environment for real-time, distributed systems. The principal investigator evaluated DCDS in terms of: a) its data collection requirements, b) its software development information completeness, c) its usability, d) how it compares to five commercially available CASE tools, and e) its suitability as an Ada Programming Support Environment (APSE).

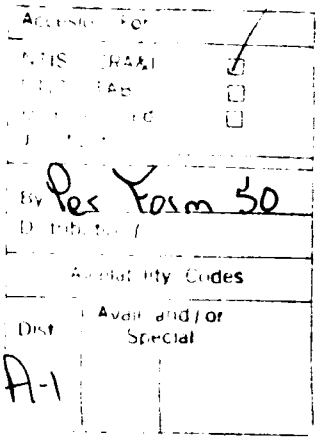
20 DISTRIBUTION / AVAILABILITY OF ABSTRACT: [X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT [ ] DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL: Howard C "Butch" Higley	22b TELEPHONE (Include Area Code): (404) 894-3110; 22c OFFICE SYMBOL: ASQB-GI

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE: UNCLASSIFIED

The research herein was performed for the Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), the RDTE organization of the U.S. Army Information Systems Engineering Command (USAISEC). The sponsor for the project was the Office of the Director of Information Systems for Command, Control, Communications, and Computers (ODISC4). The principal investigator was Dr. H. Dunsmore of Purdue University.

This research report is not to be construed as an official Army position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited, and is not protected by copyright laws. Your comments on all aspects of the document are solicited.



THIS REPORT HAS BEEN REVIEWED AND IS APPROVED

s/
Glenn E. Racine
Chief
CISD

s/
John R. Mitchell
Director
AIRMICS

Foreword

The following final technical report consists of five separate but related reports which evaluate the Distributed Computing Design System (DCDS). DCDS was developed by TRW as a software development environment for real-time distributed systems. The purpose of the project was to evaluate DCDS in terms of a) its data collection requirements, b) its software development information completeness, c) its usability, d) how it compares to five commercially available CASE tools, and e) its suitability as an Ada Programming Support Environment (APSE).

The five reports included herein are:

I. Software Development Information Supported by the Distributed Computing Design System (DCDS). Pages I-1 through I-21.

II. Software Development Information Completeness in the Distributed Computing Design System (DCDS). Pages II-1 through II-6.

III. An Evaluation of DCDS. Pages III-1 through III-24.

IV. A Usability Comparison of DCDS with Five Popular CASE Tools. Pages IV-1 through IV-7.

V. Conclusions on the Suitability of DCDS as an Ada Programming Support Environment (ASPE). Pages V-1 through V-6.

Software Development Information Supported by the Distributed Computing Design System (DCDS)

S. Varnau
H. Dunsmore

Software Engineering Research Center (SERC)
Department of Computer Sciences
Purdue University, West Lafayette, IN 47907

SERC-TR-85-P
January, 1991

Technical Report 3.3 from the Research Project:
Evaluation of DCDS for Meeting the Data Collection
Requirements for Software Specification, Development, and Support

Abstract

In this paper we examine the Distributed Computing Design System (DCDS) and evaluate it according to data collection requirements for CASE systems that we have previously developed. Our data collection requirements categorize and define what information a standard CASE environment needs to collect.

DCDS scored extremely well on our data collection criteria. Its overall mean score of 3.2 leads us to the conclusion that DCDS is an "Adequate" software development environment. This is particularly impressive when compared to the "best case" mean 2.2 for the 5 CASE tools EPOS, Teamwork, Excelerator, DesignAid, and SA Tools.

DCDS is impressive in several regards: It is flexible, with extensible languages used to build entity-relationship type databases. In addition, we were impressed with DCDS' detailed attention to software development processes. But, DCDS also has some deficiencies that we discuss in this report. Such issues will be explored more fully in the next phase of our work in which we will suggest modifications and enhancements to DCDS.

Background

In our previous work, we identified data collection requirements for CASE systems [VARN90a]. The focus of that research was to describe an environment useful and flexible enough to become a standard. Such an environment must have the flexibility to grow, evolve, and specialize. Basic to this viewpoint are portability, open architecture, and data collection. Many tools may be added to an environment, enhancing functionality, but they all need to be able to operate on the project data. Our data collection requirements categorize and define what information a standard CASE environment needs to collect.

We now evaluate the Distributed Computing Design System (DCDS) according to those requirements. DCDS consists of tools that interact with entity-relationship type databases. Each database is built using a language nucleus. DCDS has five different languages which the user may extend. He may also create his own language. Each one of the languages corresponds to a detailed methodology, which is part of the overall development process. A template processor provides the user a means to define reports and new (or modified) methodology screens.

Results

Below we present a comparison of the relative use of the data requirements identified in [VARN90a] and reprinted in the Appendix to this paper. The DCDS results are presented in conjunction with several sources surveyed earlier in this project [VARN90b]. The Henderson/Cooprider column represents data requirements identified from a report describing a functional view of CASE technology [HEND88]. The CASE tool column represents a "best case" combination of all the CASE tools studied in a previous phase of this research [VARN90a]. For each category, the score given is the maximum score of all five tools in the previous report. The best cases of those popular tools reflect current technology. The SEI questionnaire column represents data requirements needed to support modern software engineering techniques as described in the SEI questionnaire [HUMP87].

The DCDS column is the most recent data. It was obtained after the categories were refined in the earlier portions of this study. DCDS was researched more than the other sources, but every attempt was made to judge it fairly against the others. Any bias error should be no greater than the error introduced by the granularity of the rating scale used. The purpose was not to get exact measures, but to view general strengths and weaknesses from the data collection viewpoint.

The following scale is used to indicate how completely each requirement is met, or is used by each source.

- == No support at all or not addressed by tool (equivalent to 0)
1 == Possible to incorporate information, but not specifically supported
2 == Category addressed, but not fully supported
3 == Adequate
4 == Exceptional treatment of category
5 == Could not be better

Note that just because two items receive the same number for the same category, this does not mean that they are functionally equivalent for that category. For example, both the CASE Tools and the SEI Questionnaire rate a 3 for the category Description/Implementation below. This does not mean that the CASE Tools and the SEI Questionnaire have identical methods by which the developer can describe the relationships among planned and implemented components - only that we consider the Description/Implementation category as "Adequate" for both the CASE Tools and the SEI Questionnaire.

PRODUCT - Description

Category	Henderson/Cooprider	CASE Tools	SEI Questionnaire	DCDS
Functionality	3	4	2	5
Interfaces	2	4	2	4
Performance	3	4	3	4
Time Constraints	1	3	1	4
Fault Tolerances	1	2	1	4
Data Flow	3	4	1	3
Process Flow	3	4	1	4
Resources	1	3	1	4
Structure	4	4	1	4
Entity-Relation.	2	3	1	5
Communication	3	4	2	4
Data	3	3	1	4
Req./Design	3	3	3	4
Design/Perf.	3	2	2	3
Descrip./Impl.	4	3	3	4
Design/Design	1	3	2	3
Prototypes	2	2	3	3
Mean Scores	2.5	3.2	1.8	3.9
Range	1-4	2-4	1-3	3-5
Inadeq. Pctage.	41%	18%	76%	0%

Note that the *Mean Scores* average the ratings for each item (Henderson/Cooprider report, CASE Tools, SEI Questionnaire, and DCDS) for all categories. The *Range* gives an idea of the variability of that item across all categories. The *Inadeq. Pctage.* line reports the percentage of all ratings for each item that are 0, 1, or 2 (instead of 3, 4 or 5) and thus judged to be *inadequate*.

The "best case" from the 5 CASE tools we analyzed scored well in this area. This should not be surprising, because such tools generally focus on analysis and design. Even the best, however, fall short in complex areas such as fault tolerance. DCDS showed phenomenal results owing to its detailed methodology and the stringent domain of distributed, real-time computing. Functionality is fully captured in detailed requirement and specification methodologies. Entity-Relationship data is easily captured, because DCDS is built around entity-relationship-attribute type databases.

Note that the DCDS mean score of 3.9 gives it an overall rating in the product description category just below the 4 (i.e., "Exceptional") level.

PRODUCT - Implementation

Category	Henderson/Cooprider	CASE Tools	SEI Questionnaire	DCDS
Actual Product	3	3	1	3
Metrics	2	1	3	2
Library	4	2	2	3
Templates	3	3	1	3
Compile Param.	-	-	-	3
Average	2.4	1.8	1.4	2.8
Range	0-4	0-3	0-3	1-3
Inadeq. Pctage.	40%	60%	80%	20%

DCDS performs reasonably well in the product implementation categories. It is the first source we have seen that addresses the Compile Parameters category. DCDS fails to reach adequacy only for the Metrics category, but still scores better than the "best case" of other CASE tools.

PRODUCT - Verification

Category	Henderson/Cooprider	CASE Tools	SEI Questionnaire	DCDS
Test Plan	-	3	3	4
Test Tools	-	-	2	4
Test Suites	-	-	2	4
Status	-	1	3	4
Errors Found	1	2	3	4
Ver./Descrip.	-	2	2	5
Analysis	1	2	4	2
Average	0.3	1.4	2.7	3.9
Range	0-1	0-3	2-4	2-5
Inadeq. Pctage.	100%	86%	43%	14%

The SEI questionnaire is fairly strong in this area, especially stressing analysis of project data. DCDS, while not doing so well on analysis, does very well with its Test Support Methodology.

PRODUCT - Maintenance

Category	Henderson/Cooprider	CASE Tools	SEI Questionnaire	DCDS
Maintenance History	2	2	2	2
Special Cases	-	1	-	2
Complaints	-	3	1	1
Proposed Changes	-	3	1	3
General Information	-	-	-	2
Average	0.4	1.8	0.8	2.0
Range	0-2	0-3	0-2	1-3
Inadeq. Pctage.	100%	60%	100%	80%

DCDS focuses on software development. It ignores most maintenance issues.

PROCESS - Management

Category	Henderson/Cooprider	CASE Tools	SEI Questionnaire	DCDS
Schedule	4	4	3	3
Budget	-	3	3	2
Pers. Assign.	4	3	2	1
Environ. Custom.	2	2	1	4
Format Parameters	3	4	-	5
Process Plan	-	-	4	4
Average	2.2	2.7	2.2	3.2
Range	0-4	0-4	0-4	1-5
Inadeq. Pctage.	33%	33%	50%	33%

DCDS deals with traditional management issues indirectly, as development risks or constraints. Format parameters, environment customization, and emphasis on process planning are DCDS strong points. While DCDS has rich standard methodologies, they are flexible enough to be modified or even changed completely.

PROCESS - Coordination

Category	Henderson/Cooprider	CASE Tools	SEI Questionnaire	DCDS
Project Direct.	3	3	1	4
Configuration	3	3	3	1
Standards	3	-	4	3
Communication	3	1	1	1
Commun. Formats	3	-	-	1
Average	3.0	1.4	1.8	2.0
Range	3	0-3	0-4	1-4
Inadeq. Pctage.	0%	60%	60%	60%

Although DCDS maintains an elaborate set of databases, it is not a strong performer in the areas of multi-user support and version/revision control. Security is virtually not addressed by DCDS.

PROCESS - Quality Control

Category	Henderson/Cooprider	CASE Tools	SEI Questionnaire	DCDS
Quality Goals	3	-	1	3
Fault Conseq.	-	2	1	2
Target Environ.	2	-	1	3
Inspections	-	-	4	2
User Input	2	1	-	3
References	3	2	-	4
Project History	1	1	4	1
Average	1.6	0.9	1.6	2.6
Range	0-3	0-2	0-4	1-4
Inadeq. Pctage.	71%	100%	71%	43%

DCDS performs much better than the typical CASE tool in the Quality Control categories. But, it still could do much better in several such categories (e.g., project history, fault consequences, inspections, and user input).

SUMMARY

The table below summarizes the mean scores from the previous 7 tables:

MIT	CASE	SEI	DCDS	Category
2.5	3.2	1.8	3.9	Product Description
2.4	1.8	1.4	2.8	Product Implementation
0.3	1.4	2.7	3.9	Product Verification
0.4	1.8	0.8	2.0	Product Maintenance
2.2	2.7	2.2	3.2	Process Management
3.0	1.4	1.8	2.0	Process Coordination
1.6	0.9	1.6	2.6	Process Quality Control
1.9	2.2	1.8	3.2	Mean Scores

The summary table above indicates that DCDS scored extremely well on our data collection criteria. Note that its overall mean score of 3.2 leads us to the conclusion that DCDS is an "Adequate" software development environment. Note that this assessment is based solely on the information collected and maintained by DCDS according to our criteria discussed in the Appendix to this paper. But, this overall mean is particularly impressive when compared to the "best case" mean 2.2 for the 5 CASE tools EPOS, Teamwork, Excelerator, DesignAid, and SA Tools. The best mean score of any these five was able to achieve on its own was EPOS' 1.6 [VARN90a].

DCDS' success against our criteria reflects its flexibility. Its database with extensible languages allows its users to store all types of data. For that reason it did not score zero in any category, and scored very few 1's. High marks in many categories are attributable to its detailed attention to software development processes.

But, DCDS is not without its problems. Some deficiencies cannot be corrected merely by adding to the database methodology languages. Such issues will be explored more fully in the next phase of our work in which we will suggest modifications and enhancements to DCDS.

References

[VARN90a] Varnau, S. and H. Dunsmore. "Software Development Information Supported by Typical CASE Tools". Software Engineering Research Center Technical Report TR-77-P. July, 1990.

[VARN90b] Varnau, S. and H. Dunsmore. "Software Development Information Supported by the SEI Contractor Assessment Questionnaire". Software Engineering Research Center Technical Report TR-78-P. July, 1990.

[HUMP87] Humphrey, Watts S. and William L. Sweet. "A Method for Assessing the Software Engineering Capability of Contractors". Software Engineering Institute Technical Report CMU/SEI-87-TR-23. Fall, 1987.

The DCDS/Ada Methodologies group in the Systems Development Division of TRW, especially John Conover, was extremely helpful to us as we researched DCDS.

APPENDIX - Data Collection Requirements for Software Specification, Development, and Support

Below we re-present the data requirements identified in our previous report [VARN90a]. These requirements are divided into two categories. **Product** data includes everything which describes the software product itself. The typical results of a software project are the requirements, specifications, design, implementation, code metrics, test plans, etc. These materials comprise product data. **Process** data includes everything which reflects the activity involved in developing and supporting the product. This includes personnel, schedule, budget, etc.

Product data is further subdivided into **description**, **implementation**, **verification**, and **maintenance** categories. Description data consists of information from the development phases commonly known as requirements, specifications, and design. This category of items serves as a plan for the product in the initial stages of a project and as documentation in later stages. Note that description data should be flexible enough to include software analysis and design information, user documents, test plans, and anything else needed. Implementation data consists of the deliverable components of a product. This includes code and documentation for the end user. Verification data consists of correctness information (typically testing information). Maintenance data consists of information used in ensuring continuing usefulness of the project after initial delivery.

Process data is subdivided into **management**, **coordination**, and **quality control** categories. Management data is used to control the project in terms of time and resources used. Coordination data is used to help personnel communicate, thus increasing quality and productivity. Quality control data is used to ensure and generally support development of a correct, robust, safe product.

Another term which appears in this report is *component*. This is a general term referring to an element of unspecified type or a group of elements. A component usually refers to a part of the deliverable product (e.g., a code module or a document). A component may also be part of a specification, design, etc. which refers to code or documents.

PRODUCT

Product Description - Planning, development, documentation of all aspects of the specific product. This is the major category that includes most of what we think of when we think of what the software does.

Functionality - What the product must do. This information should reflect the requirements and specifications for the software. It can be in a formal, semi-formal, or just a natural language format. It should include data input, data output, product behavior, and other properties such as portability and security.

Interfaces - Interaction with external systems. This information should detail what external systems are related to this software and the specific types of interactions between the software and the external systems.

Performance - Time and space that the product uses. This is information that describes the required memory and disk space for the software, along with standard (or typical) execution times. The information may be quite complicated if the software can be run in various size configurations or if execution times are varied dependent on input parameters.

Time Constraints - Real time limitations. This information outlines the time performance constraints placed on the software. This includes any partial or total constraints placed on execution times.

Fault Tolerances - Error and failure handling. This information outlines the acceptable responses of the software to "erroneous" input or to hardware failure. Such errors and failures can include exceptions, faults, and resource limitations. The information in this category can include the types of error messages that are to appear, the kinds of errors that need not be detected, and the kinds of recoveries expected from certain errors.

Data Flow - Movement of data in and out of components or stores. This information describes the way in which data moves throughout the software. It treats each component or store as a data-handling entity and describes that data that moves in and out of that entity (including what the data is, where it came from, and where it is going).

Process Flow - Execution progression of components; sequential/parallel. This information describes the software from a control flow viewpoint discussing the flow of execution in both normal and abnormal situations. It also includes sequential and parallel control flow information.

Resources - Resource usage of component; hardware considerations. Resources are any entities external to the software. This information discusses resources that either supply information to the software or receive information from the

software.

Structure - Static decomposition of components. This information conveys any logical grouping of components for any reason; for example, grouping all components that deal with the same database. There may be several static decompositions for the same software.

Entity-Relationships - Relationships among components and externals. This information includes all of the typical E-R type information, e.g., for each entity to what other entities is it related and in what manner.

Communication - Internal interfaces. Within the software how is communication accomplished? What messages (in the object-oriented sense) are communicated among the software's entities?

Data - Often now being called the **Data Dictionary**, **Data Encyclopedia**, or **Data Repository**. Data types, operations, constants, descriptions, stores, relationships, objects and classes, processes, data flows, events, states, external entities. May be related to Project Index data (see Process Coordination).

Requirements/Design - Relationships of goals and components. This information tells which requirements are related to (satisfied by) which elements of the design.

Design/Performance - Relationships of structure and performance. This information tells which elements of the design are related to the various performance constraints.

Description/Implementation - Relationships of planned and implemented components. This information links the requirements and specifications (description of the software) with the actual implementation. That is, what components implement the requirements and specifications.

Design/Design - Relationships of alternative design representations. For software with more than one design proposed, how does each relate to the other? What are the functionality and performance tradeoffs of each?

Prototypes - What prototyping activity is planned? What specific aspects of the software is to be prototyped? What will be done with the prototype? What simulations will be conducted? What experiments will be tried to test requirements, specifications, design, etc. This information, when complete, should include the prototype goals (questions the prototype is designed to answer) and results (experimentally-determined answers), as well as the actual prototype product, simulation code, etc.

Product Implementation - This is the major category that includes the actual software product (i.e., code, documents, etc.) as well as relevant information.

Actual Product - Code, Documents for end user. This is the software and documentation produced. It consists of all new (and possibly re-used) code and the text and graphics necessary to produce documentation for the software. This category is closely related to Configuration (see Process Coordination) which keeps track of versions, revisions, etc.

Metrics - Product statistics. This information consists of any and all metrics computed primarily from the software code (but possibly also from documentation or other related representations of the product). It may include (but is not limited to) such metrics as lines-of-code, size of data structure, and complexity (e.g., v(G)). Such metrics may be used for management, testing, maintenance, performance, and even quality control purposes.

Library - Globally available, re-usable components. This information contains either actual re-usable components (or some sort of pointer to them) that will be (or have been) employed in the implementation of the software. Such a library may have project, company, or even wider scope.

Templates - Outlines and examples of common components. This information contains sample components that conform to project, company, or wider standards. Such components may simply be bare-bones schema with little actual code or may be nearly complete components that require only minor modification before use in the software.

Compile Parameters - How code is compiled for testing, debugging, and (ultimately) for generating a production version. This information includes standard compilation parameters, ways of testing various versions, searching order for external components (such as re-used components), and special parameters necessary for preparing the product version.

Product Verification - This is the major category that includes all information related to testing the software (or any related activity that attempts to discover and correct errors).

Test Plan - Outline of testing process. This contains at least rudimentary information about how the software is to be tested: what types of testing procedures (perhaps formal methods) will be pursued, what tools will be used, what types of test data, what will be done about errors that are discovered, etc.

Test Tools - Custom functions for debugging and testing. This is information about the specific tools that will be (were) used for testing the software. These can include tools that are part of the CASE tool, standalone external tools, or specific test harnesses to be produced as part of the software development process.

Test Suites - Test data and expected results. This information describes specifically how test data is to be generated, how the software is to be "exercised" with this data, and how the results are to be interpreted.

Status - This information (collected during the software testing process) outlines which tests have detected the presence of an error and which tests have failed to detect the presence of an error. Obviously, it is possible to tell from this information which tests have been run (and either detected or failed to detect errors) and which tests have not been run. For regression tests, this information will tell which have been run on which versions and which revisions.

Errors Found - Errors discovered through testing; error reports. This information outlines what errors have been discovered, which have been corrected, which are planned to be corrected, and which (if any) are not planned to be corrected.

Verification/Description - This information links the requirements and specifications (description of the software) with the verification process. That is, what has been (will be) done to assure that specific requirements and specifications have been tested.

Analysis - Results of matching implementation against description (i.e., requirements and specifications). This information includes such items as types of errors, time and space performance, error and failure handling, consistency, and completeness.

Product Maintenance - This is the major category that includes all information related to the maintenance of the software, its upkeep, and support of the product in use (and perhaps even in late development stages).

Maintenance History - This information includes all actual changes made and known problems not yet corrected. It also includes information about various software releases and versions and how they differ from each other.

Special Cases - How the product is being used. How the product is being customized. This includes any release or version related information not included in the Maintenance History sub-category above due to special circumstances.

Complaints - Reported errors and their locations, problems; evaluations; replies. This information includes all requests for changes to the software based on actual errors (i.e., the software fails to meet one of its requirements).

Proposed Changes - Reported desires for new versions (including specific modifications); evaluations; replies; planned upgrades. This information includes all requests for changes to the software based on enhancements (i.e., the software meets its requirements, but it could do something even more useful for the end user).

General Information - Any other information related to the software as it is in operation; for example, (but not limited to) market penetration, customer addresses and contacts, and versions and licenses.

PROCESS

Process Management - Resource management for the software project. This is the major category that includes most of the management information pertaining to the software development process. A good CASE tool should support most of the information maintained and manipulated by good stand-alone project management tools.

Schedule - Time to finish each task. This information will include both estimates of task durations and triggering mechanisms (for those not yet completed) as well as actual start, stop, and duration times (for those tasks already completed). It will include any relevant dependency and status information, as well.

Budget - This information includes estimates of salaries, personnel costs, hardware costs, etc. (for tasks not yet completed) as well as actual salaries, personnel costs, and hardware costs (for those tasks already completed). It will include any relevant dependency and status information, as well.

Personnel Assignments - This information includes responsibilities (who is responsible for each aspect of the software development), backups (who are available to step in for those with primary responsibilities), authorities (who has read/write access to what project data), as well as individual data (experience, skills, etc.) for each member of the software development team.

Environment Customization - This information describes the environment in which this project is being developed (including how it may differ from the standard software development environment in this company). What procedures, tools, techniques, languages, management standards, coding standards, and documentation standards are being used. How text and graphics are formatted for various media. This information outlines how the software is to (does) interact with the end users. Information such as standard screen formats, standard error formats, standard input "forms" are all included in this information.

Format Parameters - Parameters for input to and output from the CASE system, including reports throughout the software life cycle that keep management informed of the progress on this software project. What reports are to be generated, what schedule is to be followed for them, are they to be manually or automatically generated, how should they look for various media.

Process Plan - What plan is to be (was) followed in developing the software. What phases are to be employed, what standards, and overall schedule. This can even include pre-project bidding and contracting information and some allowance for process improvement.

Process Coordination - This major category includes all information needed by the software development team for cooperation, communication, and organization.

Project Directory - Project, company, or environment scope directories. This information includes all linkages to people, requirements, specifications, design, code, and testing relevant to this software project. For example, in the people category it can include all personnel working on the project, personnel with previous experience on this or a similar project, personnel with consulting capabilities outside the project, etc.

Configuration - Arrangement of all product and some process data. This includes such information as (but is not limited to) software versions, revisions (history of the software), structural relationships, and control locks (overwrite protection).

Standards - Project consistency rules. This information includes all standards that are to be (were) followed during software development. Note that several other categories include some standards. In this category they are to be all collected including documentation (perhaps the most important), personnel, design, coding, messaging, and implementation standards.

Communication - Intra-group communication. This information includes names, addresses, phone numbers, e-mail addresses, and office locations of all personnel working on the project. It can also include (but is not limited to) mail aliases (mailing lists), note logs, meeting minutes, note/component relationships (i.e., topical index for notes, references).

Communication Formats - Idea communication media. This includes information on the various modes of communication among software development team members: for example, (both in-person as well as electronic versions of the following) forums, bulletin boards, brainstorming sessions, votes, etc.

Process Quality Control - This major category includes all information pertaining to quality assurance including product quality, process quality, run-time environments, and history.

Quality Goals - Criteria to measure quality. This includes information from requirements, specifications, and otherwise that can be used to assess the quality of the completed software project.

Fault Consequences - What happens if the product fails. This information describes the severity of the problems involved if the entire product or any components thereof fail to operate according to expectations.

Target Environment - How will the product be used. The software must operate within certain hardware and software constraints. This includes such information as the type of operating system, LAN operation, possible abuses, etc.

Inspections - Standards, schedules, participants, results. This includes information about what inspections are planned (or for a completed project, what inspections were conducted). It also includes information on classes, design meetings, problem resolution meetings, and informal meetings.

User Input - Customer/End-user evaluations and comments. What user input is going to be (was) collected. How is it to be used. What effect will it have on the developing and completed software product. What input will it obtain from experts in the field.

References - Miscellaneous, external references. This can include (but is not limited to) references to similar projects, projects in the same application area, projects conducted for similar hardware systems, projects developed by the same or similar software development teams, etc.

Project History - Record of changes and results of the process. This information includes all aspects of project history that is not found in Configuration (see Process Coordination). It may include (but is not limited to) project summaries, post-mortem analyses, process evaluations, and process improvement suggestions.

Software Development Information Completeness in the Distributed Computing Design System (DCDS)

S. Varnau
H. Dunsmore

Software Engineering Research Center (SERC)
Department of Computer Sciences
Purdue University, West Lafayette, IN 47907

SERC-TR-86-P
January, 1991

Technical Report 3.4 from the Research Project:
Evaluation of DCDS for Meeting the Data Collection
Requirements for Software Specification, Development, and Support

Abstract

In this paper we explore the implications of our previous evaluation of the Distributed Computing Design System (DCDS) according to our data collection requirements. That evaluation gave insight to basic support functionality as well as data collected.

A strength of DCDS is in the extensible data model implemented in entity-relationship databases. A weakness lies in the separation of the databases. Configuration management support is also weak. DCDS supports a partially open architecture and could be enhanced to include a fully open architecture and necessary database support. Additional work could be generalization of graphics and improvement of the entire user interface. More extensive help text, examples, and better manuals would be valuable. Enhancements of lesser importance are generalization of the simulation tool and extension of methodology languages.

Our data collection evaluation has shown that DCDS is well designed to serve as the basis for a programming support environment, especially if some enhancements can be made to it. Functionality and usability issues will be discussed further in the next step of our work.

Background

In our previous work, we identified data collection requirements for CASE systems [VARN90a]. The focus of that research was to describe an environment useful and flexible enough to become a standard software project support environment. Such an environment must have the flexibility to grow, evolve, and specialize. Basic to this viewpoint are portability, open architecture, and data collection. Of these, data collection seems to be the least explored and potentially the most important. Many tools may be added to an environment, enhancing functionality, but they all operate on the same project data. Our data collection requirements categorize and define what data a standard CASE environment needs to collect. An effective data model should be fairly comprehensive over the data requirements, and have sufficient flexibility to evolve along with the environment and projects that are developed using that environment.

In the process of defining and refining the CASE data requirements, we evaluated two important papers in the CASE field and several common CASE tools [see VARN90a, VARN90b]. We then evaluated the Distributed Computing Design System (DCDS) according to those requirements [VARN90c]. Now, we use this information to determine the completeness of DCDS from the data collection viewpoint. Furthermore, we propose enhancements and modifications that will make DCDS more useful as a standard development environment.

General Discussion of DCDS

DCDS consists of tools that interact with entity-relationship-attribute (ERA) type databases. Each database is built using a language nucleus, which defines the types of allowable entities, attributes, relationships, and qualifiers. DCDS has five different languages which the user may extend, or he may create new languages. Each one of the languages corresponds to a detailed methodology, which is part of an overall development process. A template processor provides the user a means to define reports and new (or modified) methodology screens.

DCDS has succeeded in separating the software development methodology from the environment itself. The core of the environment consists of tools to create, modify, and query project databases. Development methodologies are implemented by database languages, templates, and query files. This design allows great flexibility of methodologies that may be used and of data that may be collected. Our data collection evaluation took this flexibility into consideration as well as the standard DCDS

methodologies.

A robust data model must incorporate data in specific, machine-readable formats. Tools adhering to these formats may be integrated with the environment. The environment is then responsible for certain basic functions controlling the database. While the flexibility of the DCDS database allows data of any kind to be collected, some essential, basic functions are missing from the environment. These problems were revealed during our data collection evaluation, but are not primarily data collection problems. This type of problem helps to identify the major enhancements needed in DCDS. Other deficiencies may be remedied simply by extending the languages and methodologies provided. This can be accomplished even at the user level.

Evaluation and Enhancements

The details of our data collection evaluation are contained in [VARN90c].

DCDS scored very well in most Product categories, especially Product Description and Product Verification. The standard DCDS methodologies are fairly comprehensive from the system requirements phase all the way through the testing phase. We do question whether these methodologies could be extended to cover maintenance issues more fully. Also, DCDS concentrates on real-time, embedded systems, so facilities for description, prototyping, and implementation of human interfaces are underdeveloped.

DCDS scored somewhat worse in many Process categories. In the Process Management area capabilities for handling information related to personnel, budget, and schedules need to be enhanced. In Process Coordination provisions need to be made for notes, meetings, and other communication information. Another problem in this category is lack of support for configuration issues, including version/revision control and multi-user support. The Process Quality Control category could be improved by adding process reviews and evaluations to the database. Project history data could also be represented better.

Most of the deficiencies found in our data collection evaluation are offset by the flexibility of the DCDS design. The standard languages and methodologies can be extended by the DCDS designers, other vendors, or even users without touching the core of the environment. Some problems appear not to be so simple; they will likely

require enhancements to the core environment. These include multi-user support, cross-database traceability, security, and version/revision control.

These database support issues are of primary concern; no add-on tools can make up for deficiencies here. Configuration management functions (security, version/revision control) should not be too difficult to incorporate into the DCDS environment. The technology is available. Management of multiple databases and multi-user support, however, may pose some difficult problems. Unless the divisions between databases can be made virtually transparent, traceability and maintainability will be difficult. It is possible that the database organization should be redesigned.

DCDS is designed well for flexibility and customization, but that power is not amply reflected in its documentation. A system programmer should have full access to methodology templates and query check files. Also, this flexibility must be extended to freely incorporate new tools in addition to DCDS tools.

The user interface is fair, with **much** potential for improvement. A graphical interface to the entity-relationship databases would be a big improvement, and we understand that such is already under development. Currently graphic capabilities only show decomposition attributes (e.g., F_NETs and R_NETs) of particular entities. Intuitively, this seems to be insufficient. Components of these "NETs" should be stored in the databases as ordinary entities. A generic graphics editor should be used for all ERA items, in keeping with the goals of flexibility and extensibility required of a standard format. Perhaps graphical templates could be used, much as textual templates are now.

The simulation functions of DCDS are good, but are restricted to modelling based on specific data constructs in two of the standard methodology languages (SSL and RSL). In the future, the DCDS simulation tool could be generalized in the same way as suggested for "NET" items above. Simulation templates would allow this feature to grow with the rest of the environment. Certainly, simulation is a nice function, but the database problems and even interface enhancements should take precedence.

Some valuable future enhancements could also be added to the data languages. More project management and process management need to be added. Also, a design language should be developed that makes use of an object-oriented perspective. Over time, design languages for several different domains could be developed.

Summary

Our data collection evaluation has shown that DCDS is well designed to serve as the basis for a programming support environment, especially if some enhancements can be made to it. We believe that many of these enhancements are already under development. Functionality and usability issues will be discussed further in the next step of our work.

A strength of DCDS is in the extensible data model implemented in entity-relationship databases. A weakness lies in the separation of the databases. Configuration management support is also weak. DCDS meets many of our data requirements and supports a partially open architecture. DCDS could be enhanced to include a fully open architecture and necessary database support to provide an excellent foundation software engineering environment.

Once these "necessary" improvements are made, additional work could be generalization of graphics and improvement of the entire user interface. More extensive help text, examples, and better manuals would be valuable.

Finally, enhancements of lesser importance are generalization of the simulation tool and extension of methodology languages.

References

[VARN90a] Varnau, S. and H. Dunsmore. "Software Development Information Supported by Typical CASE Tools". Software Engineering Research Center Technical Report TR-77-P. July, 1990.

[VARN90b] Varnau, S. and H. Dunsmore. "Software Development Information Supported by the SEI Contractor Assessment Questionnaire". Software Engineering Research Center Technical Report TR-78-P. July, 1990.

[VARN90c] Varnau, S. and H. Dunsmore. "Software Development Information Supported by the Distributed Computing Design System". Software Engineering Research Center Technical Report TR-85-P. January, 1991.

The DCDS/Ada Methodologies group in the Systems Development Division of TRW, especially John Conover, was extremely helpful to us as we researched DCDS.

An Evaluation of DCDS

S. Varnau
H. Dunsmore

Software Engineering Research Center (SERC)
Department of Computer Sciences
Purdue University, West Lafayette, IN 47907

SERC-TR-87-P
January, 1991

Technical Report 3.5a from the Research Project:
Evaluation of DCDS for Meeting the Data Collection
Requirements for Software Specification, Development, and Support

Abstract

This report evaluates DCDS (Distributed Computing Design System) from a usability perspective rather than our standard data collection viewpoint. The evaluation tool used was developed in an earlier SERC project.

DCDS has several major strengths: including a flexible methodology, full life cycle coverage, a very detailed methodology, and a relationship-oriented database. DCDS also has some major weaknesses: including a lack of multiple user support, a lack of database security, a lack of project management support, and a mediocre user interface.

The evaluation tool that was used to generates this report has been used on other CASE tools in the past. A comparison of DCDS and those other tools will be presented in a companion report TR 3.5b.

Background

In previous work, we evaluated DCDS and five other CASE products on the basis of the data they collected [VARN90a, VARN90b]. We now evaluate DCDS on functional, performance, and usability criteria as presented in [ZAGE87]. The evaluation questions are based on the principles of "portability, generality, tailorability, extensibility, uniformity, controlability, efficiency, reusability, and traceability". These are certainly desirable goals, but are not directly reflected in our data collection requirements [VARN90a]. The two evaluation techniques are not mutually exclusive. It is our intention that the information requirements analysis combined with this pragmatic questionnaire assessment will present a complete picture of DCDS.

This usability questionnaire has already been applied to the five other CASE products discussed in [VARN90a]. A comparison of DCDS and those other tools will be presented in a companion report TR 3.5b.

I. SYSTEM OVERVIEW

1. ***System identification and miscellanea*** ---

   a. What is the name of the system?

      Distributed Computing Design System

   b. What is its acronym (if any)?

      DCDS

   c. What version is this?

      DCDS/Ada (SUN) Release 4.3

   d. What company sells or what university developed this system?

      TRW

   e. What is the company's (university's) address?

      TRW/Huntsville
      213 Wynn Drive
      Huntsville AL 35805

f. How long has this system been on the market or how long has it been in development?

DCDS has evolved from work started in 1968. DCDS/Ada development was started in 1984.

g. How many copies have been sold?

This is unclear, but we believe it is less than 100.

h. What enhancements are planned for the future?

TRW plans to port the system to X-windows interface, add a graphic interface for general entities and relationships, and enhance the entire user interface. Dataflows will be added to F_NET graphics. Also, the document generation facility will be extended.

2. *Area of product concentration ---*

a. What is the system's concentration area (if any) (embedded systems, real-time control, system software, business software, engineering/scientific, personal computer, artificial intelligence, ...)?

Real-time, distributed systems.

3. *Hardware requirements ---*

a. On what machines (micros, workstations, mainframes) does this system run? What operating system is required in each case?

VAX 7800 or above, VMS 5.2 or later
SUN 3 family, SUN O/S 4.0 or later

b. What are the memory requirements?

VAX: 8192K, at least 32K paging file quota
SUN: 8MB, 30MB swap space

c. What are the disk (i.e., online permanent storage) requirements?

VAX: 80K Blocks
SUN: 25MB

d. What are the graphics/screen requirements?

VAX: (optional) Tektronix 4105 Terminal or TGRAF Emulator
SUN: Normal SUN 3 Hardware, runs under Suntools

e. What additional hardware/software is needed for communication with other systems?

Optional (for Simulation):

VAX: VAX/Ada Compiler
SUN: ALSYS Ada Compiler

f. Can it run in a degraded form? (For example, some features might need graphics capability. But, can users without graphics devices at least access some of these features in a usable form?)

The graphics are not needed on the VAX version.

g. How easy is it to install this system?

Easy. The executable system is delivered on one magnetic tape.

4. ***Vendor support ---***

a. What training (e.g., workshops) does the vendor offer?

There is a one week course, including lab sessions.

b. What off-line documentation (usually manuals) does the vendor offer?

User's Guide
Methodology Guide
Methodology Guide Appendices
Technical Overview

c. What on-line documentation (usually help facilities) does the vendor offer?

Context sensitive tool (general) Help and Methodology Help buttons can be selected by mouse. Methodology help templates may be enhanced by users. But, the quality of these is inconsistent.

d. Does the vendor have a toll-free number for assistance?

No. User support and training must be provided by developers.

e. What is the vendor's market position (i.e., top 10%, lower 25%, ...)?

This question is not applicable because the product is government-owned. The product is not used widely, but few products compete in the defense, real-time, distributed software arena.

f. Has the vendor made a commitment to support and enhance the system?

Since development is dependent on military financial support, the commitment is not assured, but seems to be steady at the present time.

5. ***Cost*** **---**

a. What is the system's initial purchase price?

The product is free since it is government owned.

b. What discounts are available?

N/A

c. Is the price per site or per machine?

N/A

d. What will be cost of training provided by the vendor?

Training is arranged by the Strategic Defense Command (SDC) in Huntsville, AL. To the best of our knowledge it appears that the training is free.

e. What will be cost of training done locally (i.e., in-house)?

N/A

f. What is the cost of upgraded system versions (if any become available)?

N/A

g. What are the potential local maintenance costs (if any)?

Extensive local customization is possible, but none is required. Costs will depend on what customization is done.

6. *Performance ---*

(based on literature)

a. Does the literature suggest that the development time of a project is decreased through the use of the system?

Only that it makes it possible to complete large projects on schedule.

1. How much?

N/A

2. In what part of the development cycle does it promise to help?

Completeness and consistency in requirements, design, and testing.

b. What have you found out about the shortcomings of this system?

Mainly, lack of multi-user support and lack of version/revision management features. We consider these features necessary for large systems software development that DCDS intends to support. Also, the user interface is sometimes cumbersome to use, including some actual errors. (In one graphics tool, the user can be faced with a situation in which it is impossible to continue, short of killing the DCDS process.)

II. SYSTEM APPEARANCE

7. *Presentation ---*

a. Does the system use

1. graphics?

Yes

2. color?

Yes

3. animation?

No

4. windows?

Yes

5. menus?

Yes

b. How is text entered ?

By keyboard, cut and paste, selection menu, command file, and external file references.

c. Are there input alternatives to the keyboard (e.g., mouse light pen, scanner, spoken, ...)?

Mouse

d. Does the system support both expert and novice modes?

Yes. In the sense that many commands may be entered on a command line, or by mouse and menus.

e. Is the interface customizable? (For example, does it provide facilities such as menu generators to assist in tailoring and extending the user interface?)

The tool menus are not customizable, but all methodology screens, reports, and check files are completely customizable.

8. ***Ease of use ---***

a. Has the system been "human factor" engineered to make it easy to use?

DCDS is not especially good in this regard. Several things are annoying, such as the confirmation "DONE" button that requires switching between keyboard and mouse when the "Return" key would be sufficient. Many actions invoke new windows which must be confirmed or canceled. This is appropriate in some places, but not in all. Methodology templates should be directly edited, rather than requiring many actions to enter text.

b. Does it have an intuitive command interface?

Yes, the commands make sense, and can be entered using buttons, pop-up menus, and parameter entry windows.

c. Are tasks which are done throughout the lifecycle invoked in the same manner throughout?

Yes.

d. Have you found any references concerning the ease of use of the system?

No recent ones. Several studies have looked at older VAX versions.

III. TOOL SET

9. ***Tool set provided ---***

a. Does the system enforce a single software development methodology or support several optional methodologies?

DCDS supports its own detailed methodology (divided into SYSREM, SREM, DDM, MDM, and TSM), which is basically the waterfall model with elements of spiral and prototyping models added. The user can define a new methodology or modify the standard one.

b. Are the tools integrated in the sense that information entered to any one is effectively available to the others?

Each of the five sub-methodologies has its own database language. The databases are not compatible, but all of the primary tools (except simulation) are compatible with each type of database.

c. Are the tools integrated in the sense that movement between them is easily done?

Moving from one tool to another is not especially easy; a return to the main option window is required each time. Also, a user cannot have two tool windows in view at the same time.

d. Does the environment appear unified (i.e., do the tools seem to work together)?

Yes, although there is not much interaction between tools.

e. Can you select one tool independent of the others?

The user interacts with only one tool at a time, but they are all part of the DCDS environment.

f. Can the tools be adapted (i.e., customized) by the user?

While the the life cycle methodology is completely customizable, the

tools themselves are not. The only things that can be changed are graphics colors, command and message logging, and output destinations.

g. Do the tools for program development include

1. screen generators?

No.

2. report generators?

No.

3. system generators?

Not really, but an executable model may be constructed during the design phase. The system has a simulation generator for use in simulation experiments.

4. fourth generation languages?

No.

5. code generators?

No.

h. Does it produce project documentation automatically?

Yes, using the query tool combined with report templates. Graphics are output separately in postscript format.

IV. LIFE CYCLE VIEW

10. ***Life cycle support*** **---**

a. Does the system support the entire development of the project?

Yes, although it is somewhat lacking in maintenance support.

b. Can comments (text) be entered throughout development?

Yes, all database items can be commented.

c. Does the literature for this system combine the requirements phase and the specifications phase as one? If yes, just answer part d and skip part

e.

Yes, although requirements can be designated as "originating requirements" or as "derived requirements".

d. Can the system be used during the requirements phase of a project?

Yes, DCDS includes a system requirements methodology (SYSREM) as well as a software requirements methodology (SREM). Below, we will discuss only the software requirements phase, although the system requirements phase is similar.

1. What methods are used for requirements analysis?

   Graphics (R_NETs and L_NETs), dataflow analysis of functional requirements.

2. Does it illustrate project interrelationships?

   Yes, with its entity relationship database structure. This is a strength of DCDS.

3. Does it perform consistency and completeness checks on the requirements?

   Yes, four consistency check files are available, and the user can add his own.

4. Does it produce requirements documents on demand?

   Report templates are available that specify the format of database query results in government or non-government formats. Also, the user may provide his own.

5. Is it capable of interfacing with other tools used in development?

   At the end of the requirements phase, export files may be created to establish requirement elements in other databases.

   a. Which tools?

      Common data element types may be ported to design, implementation, and test phase databases.

e. Can the system be used during the specifications phase of a project?

Yes, see above section.

1. What methods are used for specifications analysis?

N/A

2. Does it perform consistency and completeness checks on the specifications?

N/A

3. Does it produce specifications documents on demand?

N/A

4. Is it capable of interfacing with other tools used in development?

N/A

a. Which tools?

N/A

f. Will it allow for the rapid development of prototypes of the system?

Yes, but only simulations for timing analysis and executable models of design, not user interfaces.

1. Does it quickly put up screens for user review?

No.

2. Does it produce mock-up reports?

No.

3. Does it have the ability to quickly change the prototype at the user's request?

No.

4. Does the system produce an initial version of the system from the system requirements?

   No.

g. Can the system be used during the design phase of a project?

   Yes, the standard methodology is DDM (Distributed Design Methodology). MDM (Module Development Methodology) also includes design, but it is discussed in the next section.

   1. What are the design tools provided?

      a. graphical design

         Logic diagrams (L_NETs)

      b. tabular design

         None

      c. programming design language

         DDL (Distributed Design Language)
         Ada Program Design Language (L_NETs)

      d. Are these integrated?

         Somewhat. Graphic structures are represented by special methodology language constructs and referred to by other data items. But, their composition is not really represented in DDL or any of the other database languages.

   2. What methodologies are supported?

      a. data flow oriented

         Performs data flow consistency analysis. Also checks that all data is produced and used.

      b. data structure oriented

         None

c. object oriented

None. Although it could be argued that the entity relationship database promotes this kind of approach.

3. Does it perform consistency and completeness checks on the design?

Yes, five different check files are available, and the user can add his own.

4. Does it produce design documents on demand?

Report templates are available that specify the format of database query results in government or non-government formats. Also, the user may provide his own.

5. Is it easy to modify the design?

Not really, most of the design information is textual.

6. Is it capable of interfacing with other tools used in development?

Data elements may be incorporated from the requirements database, and at the end of the design phase, export files may be created to establish design elements in other databases.

a. Which tools?

Common data element types may be ported to implementation and test phase databases.

h. Can the system be used during the coding phase of a project?

Yes, the standard methodology is MDM (Module Development Methodology). MDM is for low-level design and implementation.

1. How does the system help during coding (e.g., syntax-directed editors,...)?

Only by keeping track of all requirements, design, coded modules, and other development information. DCDS also provides consistency check files for this phase. L_NETs can be used for actual coding as well as design language.

2. Does the system support automatic code generation?

No.

a. What items are needed (e.g., specifications, design, ...)?

N/A

b. In what languages can code be generated?

N/A

c. Is the generated code portable?

N/A

d. Is the generated code reusable?

N/A

e. What percentage of code can actually be generated (vs. that supplied by the user)?

N/A

i. Can the system be used during the testing phase of a project?

Yes, the standard methodology is TSM (Test Support Methodology).

1. Does the system have run-time performance tuning tools available?

No, only simulation.

2. What testing tools are provided? (i.e. test data generators, static analyzers, ...)

The Process Construction Manager (PCM) is used to build the required configurations for tests. L_NETs may be used to construct test drivers.

3. Does the system have any kind of formal verification sub-system?

No.

j. Can the system be used during the maintenance phase of a project?

Yes, all development information, including code and tests, are stored in the databases. DCDS does not, however, have a special methodology for maintenance.

1. Does the system provide the means to maintain multiple versions?

No.

2. Does the system reduce the effort for change and enhancement?

Yes.

a. How does it reduce the effort?

Access to requirements, design, implementation, and test information makes it easier to trace changes through the life cycle.

3. Does it provide documentation updates?

No.

11. *Communication vehicle(s) ---*

a. What is the communication vehicle (i.e., language) used for each stage of the development(e.g. graphics, text, VHLL)?

1. Requirements

SSL (System Specification Language), RSL (Requirements Statement Language)

Graphics (F_NETs, I_NETs, R_NETs, L_NETs)

2. Specification

RSL, DDL (Distributed Design Language)

Graphics (R_NETs, L_NETs)

3. Prototyping

None

4. Design

   DDL, MDL (Module Development Language)
   Ada Program Design Language

   Graphics (R_NETs, L_NETs)

5. Analysis

   DCDS query language and database language (SSL, RSL, DDL, MDL, TSL)

6. Coding

   Ada, or any other programming language

7. Testing

   TSL (Test Support Language)

   Graphics (L_NETs)

8. Maintenance

   Same as above languages

V. DATABASE SUPPORT

12. ***Project database ("repository of project information") ---***

    a. Does it have a central information repository?

       No, it has five central repositories! There is one for each methodology.

    b. What information is kept about each of the following in the project database?

       1. requirements

          All five languages support "ORIGINATING_REQUIREMENT", "PERFORMANCE_REQUIREMENT", "SYSTEM_REQUIREMENT", and "UNSTRUCTURED_REQUIREMENT" element types.

       2. specifications

          All five languages support the "DERIVED_REQUIREMENT"

element type.

3. design

   A great deal of the database element types are involved in the system design.

4. assumptions

   Two of the five languages support "ASSUMPTION" as an attribute of "DECISION" elements.

5. decisions

   All five languages support a "DECISION" element type.

6. modules

   MDL supports "ROUTINE" and "MODULE" element types.

7. data structure

   SSL and RSL support a "DATA" element type.

   DDL and MDL support "ACCESS_TYPE", "ARRAY_TYPE", "FILE_TYPE", "ORDINAL_TYPE", "REAL_TYPE", "RECORD_TYPE", "SET_TYPE", "SUBRANGE_TYPE", "USER_TYPE", and "VARIABLE" element types.

8. data definitions

   The attributes of the various data element types give descriptions, types, ranges, initial values, locality, units, and relationships to other elements.

9. code

   In MDL, the element types "ROUTINE" and "MODULE_VERSION" refer to the actual code.

10. test plans

    In TSL, a great many element types refer to different aspects of the test plan.

11. other

The database languages are very detailed. All data is kept within the databases directly or as an external file reference.

c. How do you retrieve the information?

Query and Text Browser are the primary retrieval tools.

d. Can you easily integrate an existing database with another project?

The query tool can be used to export data in a command format for input to DCDS, or in a text format as specified by templates. A database can also be loaded as a whole.

e. Does the system provide for completeness and consistency checking via the project database?

Yes, the standard methodologies provide many check files. The user may also change these or implement his own.

13. ***Cross referencing ---***

a. Does the system support traceability (i.e., can a requirement be traced to specifications to design to code)?

Yes. But, this is a somewhat manual process - since each phase requires a different database and only one database may be loaded at one time.

b. Does the system support cross-module indexing (i.e., can the system warn if a change in one module might necessitate a change in another)?

Yes, such relationships are captured in the database, but DCDS does not provide such a function.

14. ***Library support ---***

a. Does the system support reusability of

1. specifications?

Only through the normal export and import of database information.

2. designs?

Only through the normal export and import of database information.

3. code?

MDM contains steps to specify reusable modules.

4. test plans?

Only through the normal export and import of database information.

b. What filing and retrieval techniques are used to support reusability?

MDL element type "MODULE" refers to reusable components. These may be searched using the Query Tool or the Text Browser.

c. What OTSS (off the shelf software) (e.g., IMSL routines) is available with this system?

None

VI. REAL WORLD USABILITY

15. *Large project support ---*

a. Does the system support large projects?

DCDS documentation suggests that it is useful for large, complex projects. The system is, however, essentially a one user product.

1. Does it support multiple users?

No. BEGIN

2. Does it support concurrency control?

No

3. Does it support version control?

No

4. Does it support configuration management?

No

5. Does it provide multiple project support?

Yes, different databases may be created, loaded, backed up, and restored at will. Only one database, however, may be loaded at a time.

b. Does the system have a method for data sharing? (e.g. central database, distributed database...)

The databases can be loaded as a whole, or particular data can be exported and imported to another database.

c. What size limitations restrict large project support (e.g., maximum number of some component, maximum number of users, ...)

No known limitations.

16. ***Project management*** ---

a. Does the system provide for project management support?

Very little management support is provided. Some schedule information can be maintained in the database.

1. Is project management supported during the entire development process?

Yes

2. Does it produce measures of scope of effort required?

No

3. Does it provide measurements of progress?

No

4. Does it produce graphical and tabular reports on the progress?

No

5. Does it estimate the quality of the product produced?

No

6. Does it allow for project tracking?

   No

7. Does it produce reports on demand?

   Yes, report templates and the query tool may be used produce reports.

8. Does it compute any code metrics (e.g., control structure metrics, data structure metrics, intermodule metrics,...)

   No

17. ***Security ---***

a. Which of the following security features does the system have (access control, access logging, information encryption, others)?

   DCDS has security only to control extension to database languages. No security measures exist to control database access and modification other than UNIX file security.

VII. SYSTEM HELP

18. ***Error checking --- (e.g., from improper item deletion or alteration)***

a. Does the system provide context-sensitive checking?

   Yes, consistency check files are provided for many methodology phases. The user can even supply his own. The check files are actually query command files that search for error conditions in the database.

b. What error recovery procedures (e.g., from improper item deletion or alteration) are available?

   Only entire database backup is provided. This is a manual activity.

19. ***Expert assistance ---***

a. Does the system provide any form of expert assistance (e.g., active expert system-like support concerning ramifications of changes - modeled after "expert" human assistants)?

   No

b. Does the system highlight or (in some similar manner) present valid options?

Where possible, the user has the option to choose from a selection menu as well as type in parameters (usually element types and instances).

VIII. RELATIONSHIP WITH OUTSIDE WORLD

20. ***Integration with other systems*** ---

a. Can this system be integrated with other systems (e.g., compilers, editors, spreadsheets,...)?

No real integration capability.

b. Does the system have an open architecture (i.e., project database is readable and usable outside the system)?

The database information can be extracted in an ASCII format. The query tool and report templates may be used to obtain desired formats. It is not clear from the documentation how other tools could operate on the database.

c. Can existing software not written with this system be integrated?

It is not clear from the documentation how other tools could be called from the DCDS environment or added to tool menus.

Summary

DCDS Strengths

(1) Low cost of acquisition and training.
(2) Good support for real-time and distributed software.
(3) Full life cycle methodologies.
(4) Flexibility of methodologies.
(5) Flexible database, supporting relationship data.

DCDS methodologies are designed with complex, distributed systems in mind, but are implemented in a manner flexible enough to evolve. A system programmer or vendors could implement other methodologies easily. This philosophy, however, was abandoned for simulation and graphics tools, which are methodology dependent.

The entity-relationship database format is useful to describe relationship data, often the toughest type to capture. Good query capability allows several tools to be implemented through queries (e.g., automatic documentation and consistency checks).

DCDS Weaknesses

(1) Mediocre user interface.
(2) Poor traceability across databases.
(3) Lack of concurrent user support.
(4) Lack of version and revision control.
(5) Lack of security.
(6) Lack of project management support.

The major problem for large system builders is the lack of multiple user support, which includes concurrency and security. Even small projects would benefit from configuration management support. DCDS will never be widely used until user interface and traceability problems are corrected.

References

The DCDS/Ada Methodologies group in the Systems Development Division of TRW, especially John Conover, was extremely helpful in our research of DCDS.

[VARN90a] Varnau, S. and H. Dunsmore. "Software Development Information Supported by Typical CASE Tools". Software Engineering Research Center Technical Report TR-77-P. July, 1990.

[VARN90b] Varnau, S. and H. Dunsmore. "Software Development Information Supported by the Distributed Computing Design System". Software Engineering Research Center Technical Report TR-85-P. January, 1991.

[ZAGE87] Zage, W. M., H. E. Dunsmore, D. M. Zage, and G. Cabral. "A Tool for Evaluating Software Engineering Environments". Software Engineering Research Center Technical Report SERC-TR-2-P. June, 1987.

A Usability Comparison of DCDS with Five Popular CASE Tools

S. Varnau
H. Dunsmore

Software Engineering Research Center (SERC)
Department of Computer Sciences
Purdue University, West Lafayette, IN 47907

SERC-TR-88-P
January, 1991

Technical Report 3.5b from the Research Project:
Evaluation of DCDS for Meeting the Data Collection
Requirements for Software Specification, Development, and Support

Abstract

Evaluation criteria specified by an evaluation tool developed in 1987 represent the basis for a comparison between DCDS and five other tools. This comparison draws on the other reports that present the evaluation results for each tool.

DCDS fares very well in some respects, but is inferior in several practical aspects. This comparison provides a different perspective and a different result than our previous data requirements comparison.

Background

In previous work, we evaluated DCDS and five other CASE products on the basis of the data they collected [VARN90a, VARN90c]. We then evaluated those same tools on functional, performance, and usability criteria as set forth in [ZAGE87]. The evaluation questions are based on the principles of "portability, generality, tailorability, extensibility, uniformity, controlability, efficiency, reusability, and traceability" [ZAGE87]. These are certainly desirable goals, but are not directly reflected in our data collection requirements [VARN90a].

We now compare those latest results, focusing mainly on DCDS. Each major section of the [ZAGE87] evaluation tool is listed below, with the important differences between DCDS and the other tools.

The first goal is to further evaluate DCDS against established CASE systems. The second goal is to compare the different tool methods to identify successful techniques. This information will be useful for our final recommendations presented in the next step of our work.

As in [VARN90a], the tools used as a comparison basis are:

DesignAid version 3.55, one of the CASE 2000 tools for Computer Aided Software Engineering available from Nastec Corporation, 24681 Northwestern Highway, Southfield, Michigan 48075.

EPOS version 3.3.0 - Engineering and Project-management Oriented Support system, available from Software Products & Services (SPS), 14 East 38th Street, New York, New York 10016.

Excelerator/RTS version 1.8A, available from Index Technology Corp., One Main Street, Cambridge, Mass 02142.

SA Tools version 1.00, available from Mentor Graphics, 8500 Southwest Creekside Place, Beaverton, Oregon 97005.

Teamwork version 3.0, available from Cadre Technologies, 222 Richmond Street, Providence, Rhode Island 02903.

The five tools listed above were evaluated previously and those prior results are referred to throughout this report. The details can be found in [ZAGE88, STRA88, CABR88, ZAGE88b, CABR88b] respectively. In [VARN90e] we added to this list:

DCDS/Ada (SUN) release 4.3, available from TRW Huntsville Operations, 213 Wynn Drive, Huntsville, Alabama 35805.

System Overview

This section of the evaluations point out several major characteristics of DCDS that differentiate it from the other products. DCDS has a very low number of distributed copies compared to the others. Only Teamwork and Excelerator are aimed at real-time systems, and none claim benefits for distributed systems. DCDS was designed for both.

Three of the evaluated products are primarily personal computer products. DCDS runs on fairly powerful workstations. That constraint makes sense, considering its domain of large systems. Other products are supported much better than DCDS, but DCDS is the only free product. Cost and support are obviously conflicting goals.

System Appearance

DCDS compares favorably with other products. With selection menus and command files, text entry is more flexible in DCDS. DCDS supports novices by menus and experts by command lines. The feature, unfortunately, is not available in the methodology tool. Customization, however, is quite good in the methodology tool, but not in the other tools. The tool menus, at least, need to be customizable. Excelerator sets a good example in that area.

Like the other products, DCDS has a good command set. Some parts of DCDS are not, however, well designed for human factors. Most of the other products are much better in that respect.

Tool Set

DCDS' tools are the best in terms of methodologies and data-sharing, and among the worst in terms of multi-tasking. While DCDS supports only one methodology directly, it is the most flexible in terms of fine-tuning or major changes. The others support formal methods with little room for change.

SA Tools and EPOS are noted for utilizing central databases. DCDS is based on a similar data-sharing scheme, except the standard methodology is divided among five parts. DCDS has good data-sharing between tools, but not between methodology phases.

Teamwork has an excellent multi-tasking interface. It is possible to have two or more windows open at the same time, working with different tools. DCDS allows only one window at a time, significantly reducing productivity.

Excelerator and DesignAid include generation tools (screens, reports, and limited code). Teamwork includes fairly good code generation. DCDS has none of these tools. DCDS, along with several of the others, does provide documentation generation.

Life Cycle View

This is one of DCDS' strong points. The standard DCDS methodologies support the entire development cycle of software. EPOS is the only other of the tools that comes close.

The DCDS standard methodologies support a system requirements level as well as software requirements. Most of the other products concentrate on specification and design. Few support implementation, although some provide varying degrees of code generation. DCDS does not provide code generation, but does support implementation. Testing is the phase in which DCDS really stands out. None of the others directly support testing. Maintenance is not directly supported by any of the products, including DCDS.

Prototyping is difficult to compare. Excelerator and DesignAid lead the others in prototyping user interfaces (screens and reports). DCDS is the only tool that generates simulation experiments.

Database Support

As mentioned above in the "Tool Set" section, DCDS has a good central repository scheme. Some of the other tools have a central data dictionary, with other data decentralized (i.e., not available to all tools). Unfortunately, DCDS' database is split into five parts corresponding to each of its standard methodologies, which hampers cross-referencing.

DCDS is the only tool to include assumptions, decisions, and test plans in the database. Like SA Tools, DCDS data can include pointers to external text files. This is a very useful feature to reference outside resources and avoid tedious data entry.

Information retrieval tools are comparable to the other products. Re-use of information is somewhat difficult for lack of multi-tasking windows (as mentioned above in the "Tool Set" section). DCDS' entity-relationship database scheme relies on adding new relationships between data, not re-using data.

Real World Usability

This section covers DCDS' major problems. Multiple users, concurrency control, version control, configuration management, and security are better supported by DesignAid, EPOS, and Teamwork. These issues are particularly important because they are integral to the database. DCDS is centered around its databases, and add-on tools cannot make up these functions.

DCDS is weak in project management. But, most of the other tools do not support management any better. EPOS is the exception. This deficiency could be corrected with add-on tools and database extensions.

System Help

DCDS' command files are a useful mechanism for consistency checking. DCDS provides good context-sensitive selection menus. Unlike DesignAid and Teamwork, DCDS does not have an error recovery function.

Relationship with Outside World

None of the products studied are easy to integrate with other tools. All of the tools include some degree of data import and export.

Summary

DCDS is different from other products we have studied in its strengths and weaknesses. The unique development atmosphere and government ownership may have contributed to the marked contrast.

The basic design and flexibility integral to DCDS is far ahead of the current market. The lifecycle view and database support address software engineering problems well. These aspects were more evident in our earlier data requirements evaluation [VARN90c].

Real world usability and user interfaces often determine winners at the marketplace. DCDS is far behind others in pragmatic concerns and product "slickness" (that is, power and ease of use). DCDS is flexible enough to change in the long term, but it requires more work and problems in the short term. Users usually concentrate on the short term.

DCDS shows promise, but still has several basic problems. Our final recommendations will be given in the next step of our work.

References

[CABR88] Cabral, G., H. E. Dunsmore, S. Stratton, D. M. Zage, and W. M. Zage. "An Evaluation of Excelerator". Software Engineering Research Center Technical Report SERC-TR-15-P. May, 1988.

[CABR88b] Cabral, G., H. E. Dunsmore, S. Stratton, D. M. Zage, and W. M. Zage. "An Evaluation of Excelerator". Software Engineering Research Center Technical Report SERC-TR-16-P. May, 1988.

[STRA88] Stratton, S., W. M. Zage, G. Cabral, H. E. Dunsmore, and D. M. Zage. "An Evaluation of EPOS". Software Engineering Research Center Technical Report SERC-TR-28-P. September, 1988.

[VARN90a] Varnau, S. and H. Dunsmore. "Software Development Information Supported by Typical CASE Tools". Software Engineering Research Center Technical Report TR-77-P. August, 1990.

[VARN90b] Varnau, S. and H. Dunsmore. "Software Development Information Supported by the SEI Contractor Assessment Questionnaire". Software Engineering Research Center Technical Report TR-78-P. August, 1990.

[VARN90c] Varnau, S. and H. Dunsmore. "Software Development Information Supported by the Distributed Computing Design System". Software Engineering Research Center Technical Report TR-85-P. January, 1991.

[VARN90d] Varnau, S. and H. Dunsmore. "Software Development Information Completeness in the Distributed Computing Design System". Software Engineering Research Center Technical Report TR-86-P. January, 1991.

[VARN90e] Varnau, S. and H. Dunsmore. "An Evaluation of DCDS". Software Engineering Research Center Technical Report TR-87-P. January, 1991.

[ZAGE87] Zage, W. M., H. E. Dunsmore, D. M. Zage, and G. Cabral. "A Tool for Evaluating Software Engineering Environments". Software Engineering Research Center Technical Report SERC-TR-2-P. June, 1987.

[ZAGE88] Zage, D. M., W. M. Zage, G. Cabral, H. E. Dunsmore, and S. Stratton. "An Evaluation of DesignAid". Software Engineering Research Center Technical Report SERC-TR-19-P. September, 1988.

[ZAGE88b] Zage, D. M., W. M. Zage, G. Cabral, H. E. Dunsmore, and S. Stratton. "An Evaluation of SA Tools". Software Engineering Research Center Technical Report SERC-TR-20-P. September, 1988.

Conclusions on the Suitability of DCDS as an Ada Program Support Environment (APSE)

S. Varnau
H. Dunsmore

Software Engineering Research Center (SERC)
Department of Computer Sciences
Purdue University, West Lafayette, IN 47907

SERC-TR-89-P
January, 1991

Technical Report 3.6 from the Research Project:
Evaluation of DCDS for Meeting the Data Collection
Requirements for Software Specification, Development, and Support

Abstract

DCDS is much better suited to embedded, distributed, real-time applications than many popular CASE tools. DCDS has addressed software engineering problems across the entire software lifecycle. The design of DCDS provides much of the flexibility needed in an Ada Program Support Environment (APSE).

But, DCDS also exhibits several shortcomings. Most importantly better database support functions are necessary to provide multi-user support. The user interface also needs improvement. DCDS should provide an open architecture for greater flexibility.

DCDS is not yet suitable to serve as an APSE. While DCDS use should be encouraged to promote enhancements, mandated use is premature. Introducing DCDS as it is now might not be beneficial in all environments and could be a hindrance to some projects. After significant improvements, DCDS would certainly be an ideal foundation for a comprehensive software project environment.

Background

In our research concerning DCDS (Distributed Computing Design System) suitability as an ASPE (Ada Standard Programming Environment), we have produced six previous technical reports. We developed a list of data requirements for software engineering environments [VARN90a]. For the purposes of developing those data requirements and gaining a perspective on how well the requirements have been met in the past, we compared several CASE (Computer-Aided Software Engineering) tools and two papers related to CASE tools to our data requirements [VARN90a, VARN90b]. The next data requirements comparison rated DCDS itself [VARN90c]. That report was followed by a discussion of DCDS from a data completeness perspective [VARN90d]. While that evaluation was data-oriented, functional and non-functional requirements are also of concern. To address those issues, we utilized an evaluation questionnaire previously developed by this research group [ZAGE87]. The questionnaire was used to evaluate DCDS [VARN90e], and compare it to similar evaluations of other products [VARN90f].

We now present our final recommendations concerning the future of DCDS. We found both data-oriented and function-oriented perspectives essential in order to gain a complete understanding of CASE tools and DCDS in particular. This research and previous research has also given us firm ideas on the requirements for an ASPE or any standard environment.

Requirements for Standard Software Development Environments

While developing our data requirements, we necessarily relied on an image of an ideal standard development environment. The software engineering domain is large and complex, with many, often conflicting, goals. Two overriding goals are product quality and development efficiency. Those goals are difficult enough, but a standard environment must, above all, be flexible.

Environment flexibility is manifest in the data collected and functionality available. If a new type of project data comes into use, adding it to normal environment operations should be an easy task for an end user or system programmer. Likewise, a new tool should be easily integrated.

In light of this goal, a standard development environment need be no more than a foundation for a complete development environment, customizable for unique vendors, departments, projects, and people. Different domains require very different tools and processes, so an environment must support arbitrary tools. While it is desirable that the original foundation come with enough data models and tools to be usable as is, they should soon be customized, enhanced, or even superceded by local and third-party tools. Those applications can be changed at any time. The primary goal for the environment is to provide a solid foundation to build on, while providing a common basis for wide-spread cooperation.

Customized environments may be very different from the original and one another, but a project must be portable between environments and stable across environment upgrades. A project is represented by the product and process data it generates over time. Maintaining that data over long periods is important for several reasons. Product maintenance and process improvement are primary concerns.

In our evaluation we have included many requirements beyond a foundational environment, because a foundation must provide support for such data and functions, and all environments, including DCDS, go well beyond a foundation in some respects.

Aside from the foundational model, our requirements focus on large, complex projects. Such projects are the primary domain of software engineering. Communication, coordination, and reliability are examples of major problems. Smaller projects and environments could benefit from some of the requirements, but they are not the concern here.

DCDS as a Standard

As reported earlier [VARN90d, VARN90f], DCDS has made significant advances in software engineering environment design. Unfortunately, DCDS is not mature enough to stand alone as the primary tool in large projects. It fits our concept of a foundation environment fairly well, but fails in the area of multi-user support.

DCDS meets our data requirements quite well. It is centered around a very flexible entity-relationship database. This design provides a good way to capture relationships (usually the most difficult data to capture). Another benefit is that the data model is modifiable and is part of the project data. Flexibility and portability are both achieved.

The lack of configuration, version, and revision control are problems with the DCDS database design. These are related to the lack of multi-user support. The default data model division into five separate parts is yet another concern. Relationships across databases are quite difficult to maintain, seriously hampering traceability. It is conceivable that a complete data model could be combined into one database. Efficiency may make that impractical. Note that these database problems are foundation problems, i.e., no add-on tools can properly fix these problems.

As for functional and usability requirements, DCDS meets some well and others poorly. The user interface has some good qualities, but needs some work. The benefits of a windowing interface are nullified because the user may work with only one window at a time. The methodology screens, reports, and consistency checks may be customized by the user. These permit great flexibility, but do not provide effective selection menus and alternate command modes as the non-methodology interfaces do. The tool interfaces are fairly good, but are not customizable. They need to be flexible to allow integration of new or specialized tools.

DCDS provides some advanced functions, such as a simulation generator. These are good, but basic environment functions are more important at this point. Another factor to consider is that the graphics and simulation packages are not as general as the rest of the environment. They are dependent on the default data model. They should be adaptable to new data models. Suggestions for achieving this are included in [VARN90d].

The Future for DCDS

DCDS serves as an excellent prototype of an Ada Program Support Environment (APSE). It is certainly better in most data collection respects than other available CASE tools. DCDS lacks, however, a sleek user interface and other features now common in other products. But, DCDS database design represents a huge improvement over most. The temptation to enforce use of the best environment available should, however, be resisted. The benefits would probably not be worth the costs at this point. Standardization should be put off until further improvements and maturity of DCDS can be verified.

Introduction of any tool requires training and support overhead. A primary tool such as an environment requires a strong commitment by all participants to be successful. The tool must make all users' jobs easier, or it will not be properly utilized. Introduction of a marginal CASE environment may not produce benefits, unless

supplemented with other tools. Emphasis needs to be placed on developing a good foundation environment, and building from there. Continued improvement is dependent on solid and long-term support for DCDS and those developing it. Actual use of the environment is also very important to growth, providing valuable feedback to DCDS developers. DCDS developers should also be users, using it to support its own continued development.

DCDS needs better database support and an open architecture. All major tools should be independent of a particular data model. The user interface can be improved, providing flexibility and usability.

DCDS is not yet suitable to serve as an APSE. While DCDS use should be encouraged to promote enhancements, mandated use is premature. Introducing DCDS as it is now might not be beneficial in all environments and could be a hindrance to some projects. After significant improvements, DCDS would certainly be an ideal foundation for a comprehensive software project environment.

References

[VARN90a] Varnau, S. and H. Dunsmore. "Software Development Information Supported by Typical CASE Tools". Software Engineering Research Center Technical Report TR-77-P. August, 1990.

[VARN90b] Varnau, S. and H. Dunsmore. "Software Development Information Supported by the SEI Contractor Assessment Questionnaire". Software Engineering Research Center Technical Report TR-78-P. August, 1990.

[VARN90c] Varnau, S. and H. Dunsmore. "Software Development Information Supported by the Distributed Computing Design System". Software Engineering Research Center Technical Report TR-85-P. January, 1991.

[VARN90d] Varnau, S. and H. Dunsmore. "Software Development Information Completeness in the Distributed Computing Design System". Software Engineering Research Center Technical Report TR-86-P. January, 1991.

[VARN90e] Varnau, S. and H. Dunsmore. "An Evaluation of DCDS". Software Engineering Research Center Technical Report TR-87-P. January, 1991.

[VARN90f] Varnau, S. and H. Dunsmore. "A Usability Comparison of DCDS with Five Popular CASE Tools". Software Engineering Research Center Technical Report TR-88-P. January, 1991.

[ZAGE87] Zage, W. M., H. E. Dunsmore, D. M. Zage, and G. Cabral. "A Tool for Evaluating Software Engineering Environments". Software Engineering Research Center Technical Report SERC-TR-2-P. June, 1987.